Class 6: R functions

Sarah Mirsaidi Madjdabadi, A16890196

Table of contents

1. Function Basics

Let's start writing our first silly function to add some numbers:

Every r function has 3 things: - name (we get to pick this) - input arguments (there can be loads of these separated by a comma) - the body (the R code that does the work)

```
add <- function(x, y=10, z=0){
  x + y + z
}
```

I can just use this function like any other function as long as R knows about it (i.e. run the code chunk)

```
add(1, 100)
```

```
[1] 101
```

```
add( x=c(1,2,3,4), y=100)
```

```
[1] 101 102 103 104
```

```r
add(1)
```

```
[1] 11
```

Functions can have "required" input arguments and "optional" input arguments. The optional arguments are defined with an equals default value (`y=10`) in the function definition.

```r
add(x=1, y=100, z=10)
```

```
[1] 111
```

> Q. Write a function to return a DNA sequence of a user specified length. Call it `generate_DNA()`

The `sample()` function can help here

```r
#generate_dna <- function(size=5) {}

students <- c("jeff", "jeremy", "peter")

sample(students, size = 5, replace = TRUE)
```

```
[1] "jeff"   "jeremy" "jeff"   "peter"  "jeremy"
```

2. Generate DNA Function

Now work with `bases` rather than `students`

```r
bases <- c("A", "C", "G", "T")

sample(bases, size = 10, replace = TRUE)
```

```
 [1] "C" "T" "C" "T" "G" "T" "G" "G" "A" "T"
```

Now I have a working `snippet` of code I can use this as the body of my first function version here:

```r
generate_dna <- function(size=5) {bases <- c("A", "C", "G", "T")

sample(bases, size = size, replace = TRUE)

}
```

```r
generate_dna(100)
```

```
  [1] "G" "G" "T" "G" "T" "G" "T" "A" "C" "A" "G" "A" "A" "C" "A" "A" "C" "T"
 [19] "T" "A" "T" "A" "T" "G" "C" "G" "T" "A" "G" "A" "T" "A" "G" "G" "C" "A"
 [37] "C" "A" "A" "T" "A" "C" "C" "T" "G" "G" "T" "G" "C" "C" "G" "C" "G" "A"
 [55] "G" "A" "T" "T" "G" "T" "G" "G" "A" "G" "T" "C" "C" "A" "G" "C" "T" "G"
 [73] "C" "G" "A" "G" "T" "C" "T" "T" "C" "C" "G" "G" "G" "T" "T" "G" "G" "C"
 [91] "G" "G" "G" "A" "A" "G" "T" "C" "C" "A"
```

```r
generate_dna()
```

```
[1] "A" "A" "A" "T" "C"
```

I want the ability to return a sequence like "AGTACCTG" i.e. a one element vector where the bases are all together.

```r
generate_dna <- function(size=5, together=TRUE) {bases <- c("A", "C", "G", "T")

sequence <- sample(bases, size = size, replace = TRUE)
if(together) {
  sequence <- paste(sequence, collapse = "")
}
return(sequence)
}
```

```r
generate_dna()
```

```
[1] "ACTAT"
```

```r
generate_dna(together = F)
```

```
[1] "T" "T" "T" "T" "A"
```

3. Generate Protein Function

> Q1. Write a protein sequence generating function that will return sequences of a user specified length?
>
> Q2.Generate random protein sequences of length 6 to 12 amino acids.
>
> Q3. Determine if these sequences can be found in nature or are they unique? Why or why not?

We can get the set of 20 natural amino acids from the **bio3d** package.

```r
aa <- bio3d::aa.table$aa1[1:20]

aa
```

```
 [1] "A" "R" "N" "D" "C" "Q" "E" "G" "H" "I" "L" "K" "M" "F" "P" "S" "T" "W" "Y"
[20] "V"
```

> Q1

```r
generate_protein <- function(size=6, together=TRUE){aa
  sequence <- sample(aa, size=size, replace=TRUE)
  if(together){
   sequence <- paste(sequence, collapse = "")
  }
  return(sequence)
}
```

```r
generate_protein()
```

```
[1] "KVVLKS"
```

> Q2

We can fix this inability to generate multiple sequences by either editing and adding to the function body code (e.g. a for loop) or by using the R **apply** family utility function.

```r
sapply(6:12, generate_protein)
```

```
[1] "MYIESE"       "QDCMNQM"      "PVFMVDIF"     "NKSCCLSWD"    "RRNMEPVYNW"
[6] "WGKLHHLYEQF"  "VYVRGRYMMITR"
```

It would be cool and useful if I could get FASTA format output.

```r
ans <- sapply(6:12, generate_protein)
ans
```

```
[1] "AANMVN"      "ASDPAAH"     "TNYACIDH"    "ESVPSVHYN"   "QDEANMAEYM"
[6] "LKQPAFQKETY" "MEEWEIPIMTDI"
```

```r
cat(ans, sep="\n")
```

```
AANMVN
ASDPAAH
TNYACIDH
ESVPSVHYN
QDEANMAEYM
LKQPAFQKETY
MEEWEIPIMTDI
```

I want this to look like FASTA format with an ID line

```
>ID.6
TPEKMV
>ID.7
KLCDMKR
>ID.8
RDYMASDR
```

The functions 'paste()' and 'cat()' can help us here...

```r
cat(paste(">ID.", 7:12, "\n", ans, sep= ""), sep="\n")
```

```
>ID.7
AANMVN
>ID.8
ASDPAAH
>ID.9
TNYACIDH
>ID.10
ESVPSVHYN
>ID.11
```

```
QDEANMAEYM
>ID.12
LKQPAFQKETY
>ID.7
MEEWEIPIMTDI
```

```r
id.line <- paste(">ID.", 6:12, sep="")
id.line
```

```
[1] ">ID.6"  ">ID.7"  ">ID.8"  ">ID.9"  ">ID.10" ">ID.11" ">ID.12"
```

```r
seq.line <- paste(id.line, ans, sep="\n")
cat(seq.line, sep="\n")
```

```
>ID.6
AANMVN
>ID.7
ASDPAAH
>ID.8
TNYACIDH
>ID.9
ESVPSVHYN
>ID.10
QDEANMAEYM
>ID.11
LKQPAFQKETY
>ID.12
MEEWEIPIMTDI
```

Q3

I BLASTp searched my FASTA format sequences against NR and found that length 6, 7, 8 are not unique and can be found in the databases with 100% coverage and 100% identity.

Random sequences of length 9 and above are unique and can't be found in the databases.

BLASTp has a window of 9.